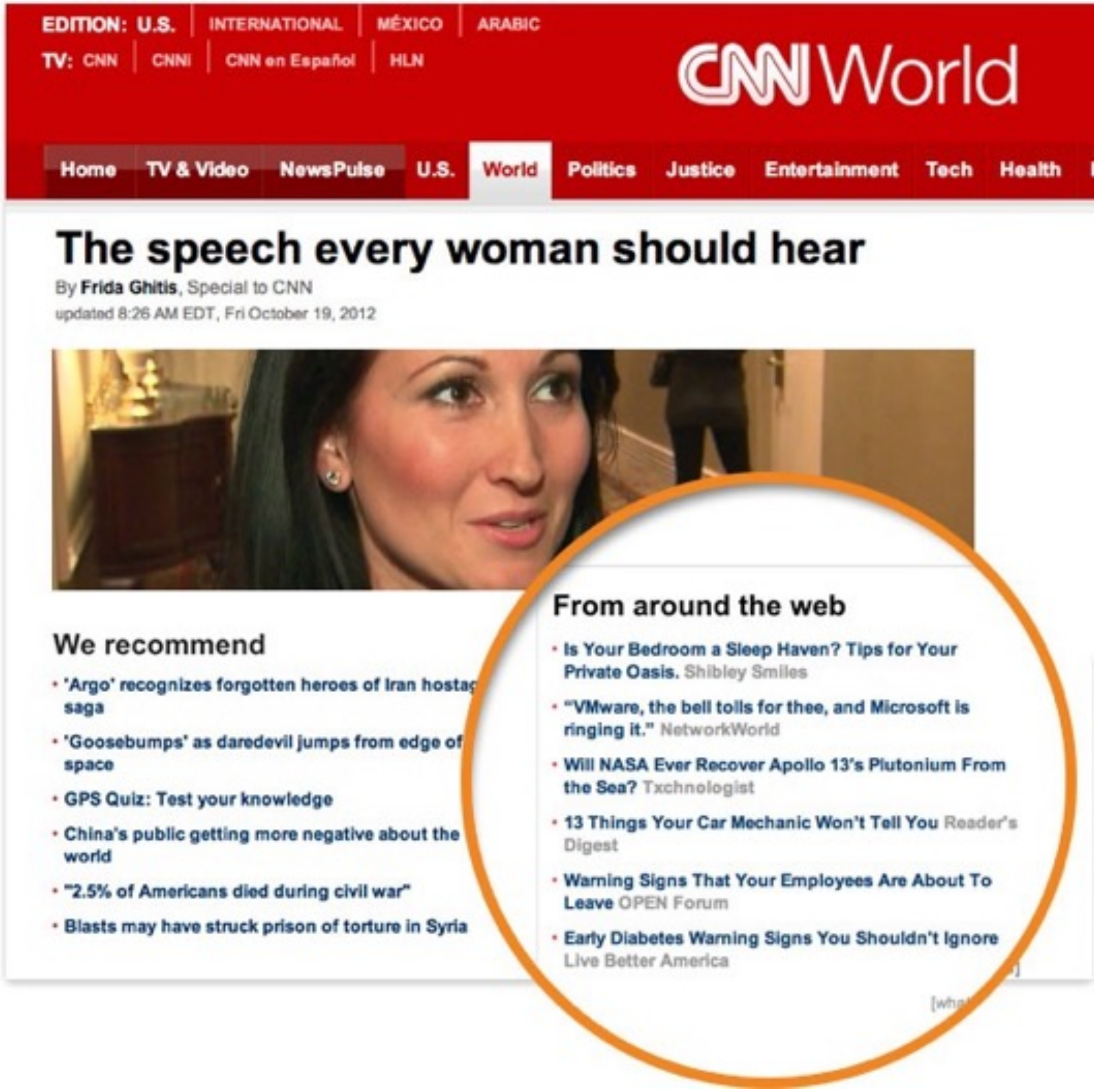
Sublinear Time Recommendation Algorithms

Vivek Farias Andrew Li Deeksha Sinha
MIT

Motivation


EDITION: U.S. | INTERNATIONAL | MÉXICO | ARABIC
TV: CNN | CNNI | CNN en Español | HLN
CNN World
Home TV & Video NewsPulse U.S. World Politics Justice Entertainment Tech Health
The speech every woman should hear
By Frida Ghitis, Special to CNN
updated 8:26 AM EDT, Fri October 19, 2012
We recommend
'Argo' recognizes forgotten heroes of Iran hostag saga
'Goosebumps' as daredevil jumps from edge of space
GPS Quiz: Test your knowledge
China's public getting more negative about the world
"2.5% of Americans died during civil war"
Blasts may have struck prison of torture in Syria
From around the web
Is Your Bedroom a Sleep Haven? Tips for Your Private Oasis. Shibley Smiles
"VMware, the bell tolls for thee, and Microsoft is ringing it." NetworkWorld
Will NASA Ever Recover Apollo 13's Plutonium From the Sea? Txchnologist
13 Things Your Car Mechanic Won't Tell You Reader's Digest
Warning Signs That Your Employees Are About To Leave OPEN Forum
Early Diabetes Warning Signs You Shouldn't Ignore Live Better America

Recommendations Today

Recommendations Today

- Two key steps
  - *Learning:* estimate a predictive model using past data
  - *Optimization:* make recommendation decisions in real-time

Recommendations Today

- Two key steps
  - *Learning:* estimate a predictive model using past data
  - *Optimization:* make recommendation decisions in real-time
- **This Talk: Operationalize existing predictive models**
  - Formulate as optimization problem
  - Propose *sublinear* time algorithm with provable guarantees
  - Show substantial improvement on massive dataset from Outbrain

Model

Model

- Products $v_1, \ldots, v_n \in \mathbb{S}^{d-1}$
- Customers are random vectors $U \in \mathbb{S}^{d-1}$

Model

- Products $v_1, \ldots, v_n \in \mathbb{S}^{d-1}$
- Customers are random vectors $U \in \mathbb{S}^{d-1}$
- Utility of clicking product $v \in \{v_1, \ldots, v_n\}$: $v^\top U + \epsilon_v$
    - γ encodes utility for 'no click'

Model

- Products $v_1, \ldots, v_n \in \mathbb{S}^{d-1}$
- Customers are random vectors $U \in \mathbb{S}^{d-1}$
- Utility of clicking product $v \in \{v_1, \ldots, v_n\}$: $v^\top U + \epsilon_v$
  - γ encodes utility for 'no click'
- Optimization problem:

$$\max_{|S| \leq k} \mathbb{P}\left(\max_{v \in S} (v^\top U + \epsilon_v) > \gamma \right)$$

Model

- Products $v_1, \ldots, v_n \in \mathbb{S}^{d-1}$
- Customers are random vectors $U \in \mathbb{S}^{d-1}$
- Utility of clicking product $v \in \{v_1, \ldots, v_n\}$: $v^\top U + \epsilon_v$
  - γ encodes utility for 'no click'
- Optimization problem:

$$\max_{|S| \le k} \underbrace{\mathbb{P}\left(\max_{v \in S} (v^\top U + \epsilon_v) > \gamma \right)}_{f(S)}$$

Concrete Example: Spotify

- Learn a mapping from 10^8 products to $\mathbb{R}^{1000}$
- Good embedding: $\|v_i - v_j\|$ small if similar vis-a-vis 'co-occurrences'
- Many algorithms do this
  - Collaborative filtering, matrix factorization [Bernhardsson '14]
  - NLP models: word2vec on playlists [Johnson '15]
  - Neural networks: CNN on audio files [Dieleman '14]
- Customers: distributions on $v_1, \ldots, v_n$

Model

- Products $v_1, \ldots, v_n \in \mathbb{S}^{d-1}$
- Customers are random vectors $U \in \mathbb{S}^{d-1}$
- Click happens if $v^\top U > \gamma$
  - γ encodes threshold for click
- Optimization problem:

$$\max_{|S| \leq k} \mathbb{P}\left(\max_{v \in S}(v^\top U + \epsilon_v) > \gamma\right)$$

Greedy Algorithm

1. Sample $\hat{U} = \{u_1, ...u_s\}$ from U
2. Sample products by potential relevance
3. Greedy algorithm over customer samples and relevant products

Greedy Algorithm

1. Sample $\hat{U} = \{u_1, ...u_s\}$ from U
2. Sample products by potential relevance
   - For each $u \in \hat{U}$, sample a subset of products $\mathcal{V}_u \subset \{v_1, \ldots, v_n\}$
3. Greedy algorithm over customer samples and relevant products

Greedy Algorithm

1. Sample $\hat{U} = \{u_1, ...u_s\}$ from U
2. Sample products by potential relevance
   - For each $u \in \hat{U}$, sample a subset of products $\mathcal{V}_u \subset \{v_1, \ldots, v_n\}$
   - $v \in \mathcal{V}_u$ with probability $\mathbb{P}\left(v^\top u + \epsilon_v > \gamma\right)$
3. Greedy algorithm over customer samples and relevant products

Greedy Algorithm

1. Sample $\hat{U} = \{u_1, ...u_s\}$ from U
2. Sample products by potential relevance
   - For each $u \in \hat{U}$, sample a subset of products $\mathcal{V}_u \subset \{v_1, \ldots, v_n\}$
   - $v \in \mathcal{V}_u$ with probability $\mathbb{P}\left(v^\top u + \epsilon_v > \gamma\right)$
   - Set $\mathcal{V} = \mathcal{V}_{u_1} \cup \cdots \cup \mathcal{V}_{u_s}$
3. Greedy algorithm over customer samples and relevant products

Greedy Algorithm

1. Sample $\hat{U} = \{u_1, ...u_s\}$ from U
2. Sample products by potential relevance
   - For each $u \in \hat{U}$, sample a subset of products $\mathcal{V}_u \subset \{v_1, \ldots, v_n\}$
   - $v \in \mathcal{V}_u$ with probability $\mathbb{P}\left(v^\top u + \epsilon_v > \gamma\right)$
   - Set $\mathcal{V} = \mathcal{V}_{u_1} \cup \cdots \cup \mathcal{V}_{u_s}$
3. Greedy algorithm over customer samples and relevant products
   - Repeat while $|S| < k$:
     - $v^* = \underset{v \in \mathcal{V}}{\operatorname{argmax}} f(S \cup v)$
     - $S = S \cup v^*$

Greedy Algorithm

1. Sample $\hat{U} = \{u_1, ...u_s\}$ from U
2. Sample products by potential relevance
   - For each $u \in \hat{U}$, sample a subset of products $\mathcal{V}_u \subset \{v_1, \ldots, v_n\}$
   - $v \in \mathcal{V}_u$ with probability $\mathbb{P}\left(v^\top u + \epsilon_v > \gamma\right)$
   - Set $\mathcal{V} = \mathcal{V}_{u_1} \cup \cdots \cup \mathcal{V}_{u_s}$
3. Greedy algorithm over customer samples and relevant products
   - Repeat while $|S| < k$:
     - $v^* = \operatorname*{argmax}_{v \in \mathcal{V}} f(S \cup v)$
     - $S = S \cup v^*$

Assumptions

1. **Bounds on Customer Consumption**

   a. **Constant Upper Bound:** For any fixed user realization u in the support of U, expected number of conversions does not grow

   $$\mathsf{E}\left[\sum_{v} \mathbf{1}\left(v^{\top} u + \epsilon_v > \gamma\right)\right] = O(1)$$

   b. **Lower Bound:** Optimal conversion does not decay fast

   $$\mathrm{OPT} = \Omega\left(\frac{1}{n^{\frac{1}{3}}}\right)$$

Assumptions

1. **Bounds on Customer Consumption**

   a. **Constant Upper Bound:** For any fixed user realization u in the support of U, expected number of conversions does not grow

   $$\mathsf{E}\left[\sum_{v} \mathbf{1}\left(v^{\top} u + \epsilon_v > \gamma\right)\right] = O(1)$$

   b. **Lower Bound:** Optimal conversion does not decay fast

   $$\mathrm{OPT} = \Omega\left(\frac{1}{n^{\frac{1}{3}}}\right)$$

2. **Good Embeddings:** Product vectors are 'uniformly spread'

$$|B_{c\gamma}(u) \cap V| = O(c^d)|B_{\gamma}(u) \cap V|$$

An (Impractical) Result

- **Theorem [Farias, Li, S.]:** Under the stated assumptions, the greedy algorithm achieves in expectation

$$(1-\delta)(1-e^{-1})\mathrm{OPT}$$

An (Impractical) Result

- **Theorem [Farias, Li, S.]:** Under the stated assumptions, the greedy algorithm achieves in expectation

$$(1-\delta)(1-e^{-1})\mathrm{OPT}$$

with running time

$$\tilde{o}_{\delta}\left(kn^{\frac{4}{3}}\right)$$

An (Impractical) Result

- **Theorem [Farias, Li, S.]:** Under the stated assumptions, the greedy algorithm achieves in expectation

$$(1-\delta)(1-e^{-1})\mathrm{OPT}$$

with running time

$$\tilde{o}_\delta\left(kn^{\frac{4}{3}}\right)$$

An (Impractical) Result

- **Theorem [Farias, Li, S.]:** Under the stated assumptions, the greedy algorithm achieves in expectation

$$(1-\delta)(1-e^{-1})\mathrm{OPT}$$

with running time

$$\tilde{o}_{\delta}\left(kn^{\frac{4}{3}}\right)$$

Goal: Sublinear time recommendation algorithm

Greedy Algorithm

1. Sample $\hat{U} = \{u_1, ...u_s\}$ from U
2. Sample products by potential relevance
   - For each $u \in \hat{U}$, sample a subset of products $\mathcal{V}_u \subset \{v_1, \ldots, v_n\}$
   - $v \in \mathcal{V}_u$ with probability $\mathbb{P}\left(v^\top u + \epsilon_v > \gamma\right)$
   - Set $\mathcal{V} = \mathcal{V}_{u_1} \cup \cdots \cup \mathcal{V}_{u_s}$
3. Greedy algorithm over customer samples and relevant products
   - Repeat while $|S| < k$:
     - $v^* = \underset{v \in \mathcal{V}}{\operatorname{argmax}} f(S \cup v)$
     - $S = S \cup v^*$

Greedy Algorithm

$\hat{U} = \{u_1, ...u_s\}$ U

2. Sample products by potential relevance

- For each $u \in \hat{U}$, sample a subset of products $\mathcal{V}_u \subset \{v_1, \ldots, v_n\}$
- $v \in \mathcal{V}_u$ with probability $\mathbb{P}\left(v^\top u + \epsilon_v > \gamma\right)$
- Set $\mathcal{V} = \mathcal{V}_{u_1} \cup \cdots \cup \mathcal{V}_{u_s}$

3. Greedy algorithm over customer samples and relevant products

- Repeat while $|S| < k$:
  - $v^* = \operatorname*{argmax}_{v \in \mathcal{V}} f(S \cup v)$
  - $S = S \cup v^*$

The Sublinear Sampling Problem

The Sublinear Sampling Problem

- "Sample $\mathcal{V}_u \subset \{v_1, \ldots, v_n\}$ s.t. $v \in \mathcal{V}_u$ w.p. $\mathbb{P}\left(v^\top u + \epsilon_v > \gamma\right)$"

The Sublinear Sampling Problem

- "Sample $\mathcal{V}_u \subset \{v_1, \ldots, v_n\}$ s.t. $v \in \mathcal{V}_u$ w.p. $\mathbb{P}\left(v^\top u + \epsilon_v > \gamma\right)$"
- Input:
  - Set of vectors $\{v_1, \ldots, v_n\}$ with norm $\|\cdot\|$
  - Query point u from same normed space
  - Membership probability $p : \mathbb{R}^+ \to [0, 1]$

The Sublinear Sampling Problem

- "Sample $\mathcal{V}_u \subset \{v_1, \ldots, v_n\}$ s.t. $v \in \mathcal{V}_u$ w.p. $\mathbb{P}\left(v^\top u + \epsilon_v > \gamma\right)$"
- Input:
  - Set of vectors $\{v_1, \ldots, v_n\}$ with norm $\|\cdot\|$
  - Query point u from same normed space
  - Membership probability $p : \mathbb{R}^+ \to [0, 1]$
- Output: $\mathcal{V} \subset \{v_1, \ldots, v_n\}$ such that

$$\mathbb{P}(v_j \in \mathcal{V}) = p(\|v_j - u\|)$$

- Goal: construct $\mathcal{V}$ in sublinear (amortized) runtime

Our Result

Our Result

- **Theorem [Farias, Li, S.]:** Under the stated assumptions, the *locality-sensitive sampling* algorithm approximates any membership probability function s.t.

$$(1-\epsilon)\cdot p(\|v_j-u\|) \leq \mathbb{P}(v_j \in \mathcal{V}) \leq (1+\epsilon)\cdot p(\|v_j-u\|)$$

Our Result

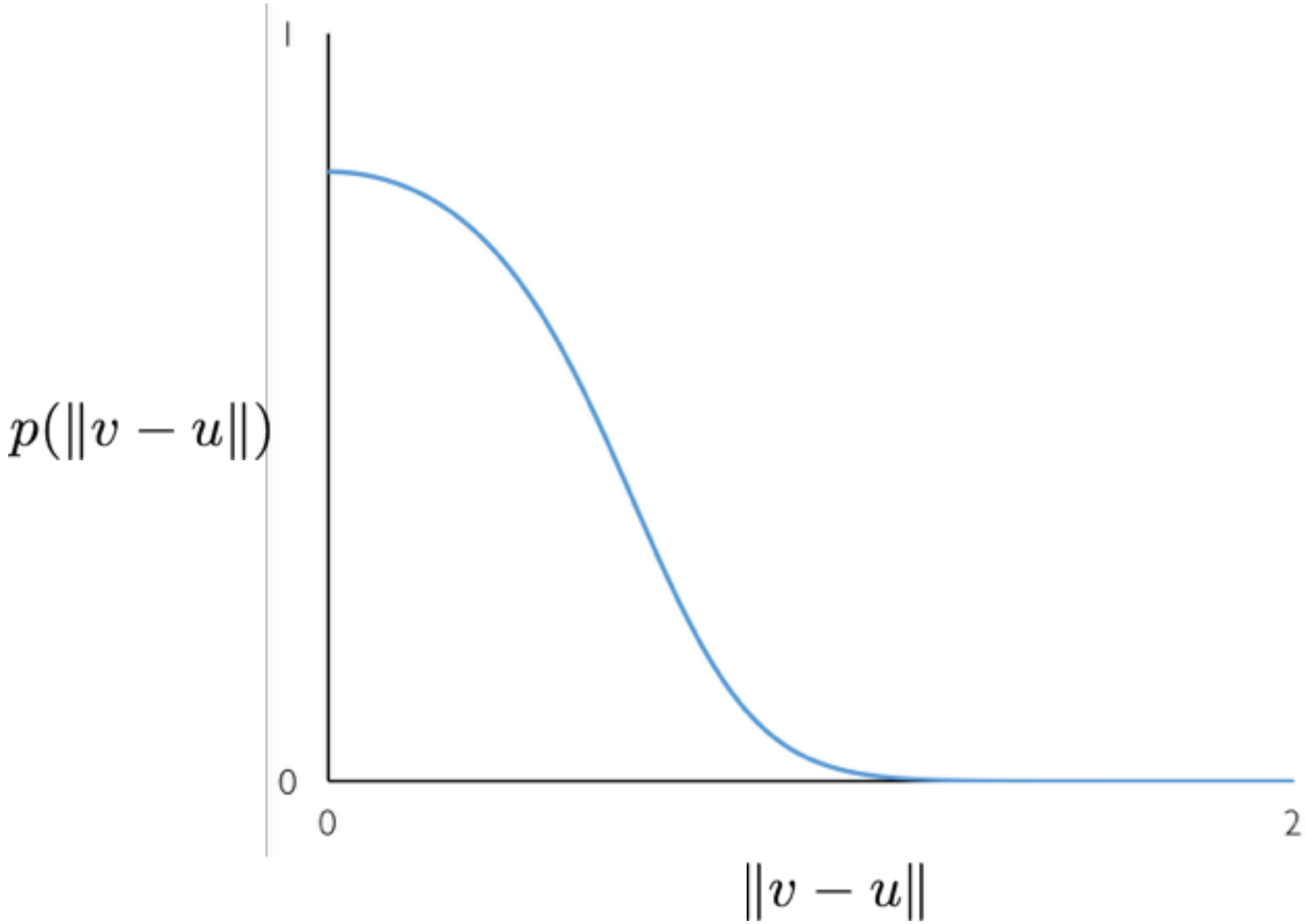
- **Theorem [Farias, Li, S.]:** Under the stated assumptions, the *locality-sensitive sampling* algorithm approximates any membership probability function s.t.

$$(1-\epsilon)\cdot p(\|v_j - u\|) \leq \mathbb{P}(v_j \in \mathcal{V}) \leq (1+\epsilon)\cdot p(\|v_j - u\|)$$

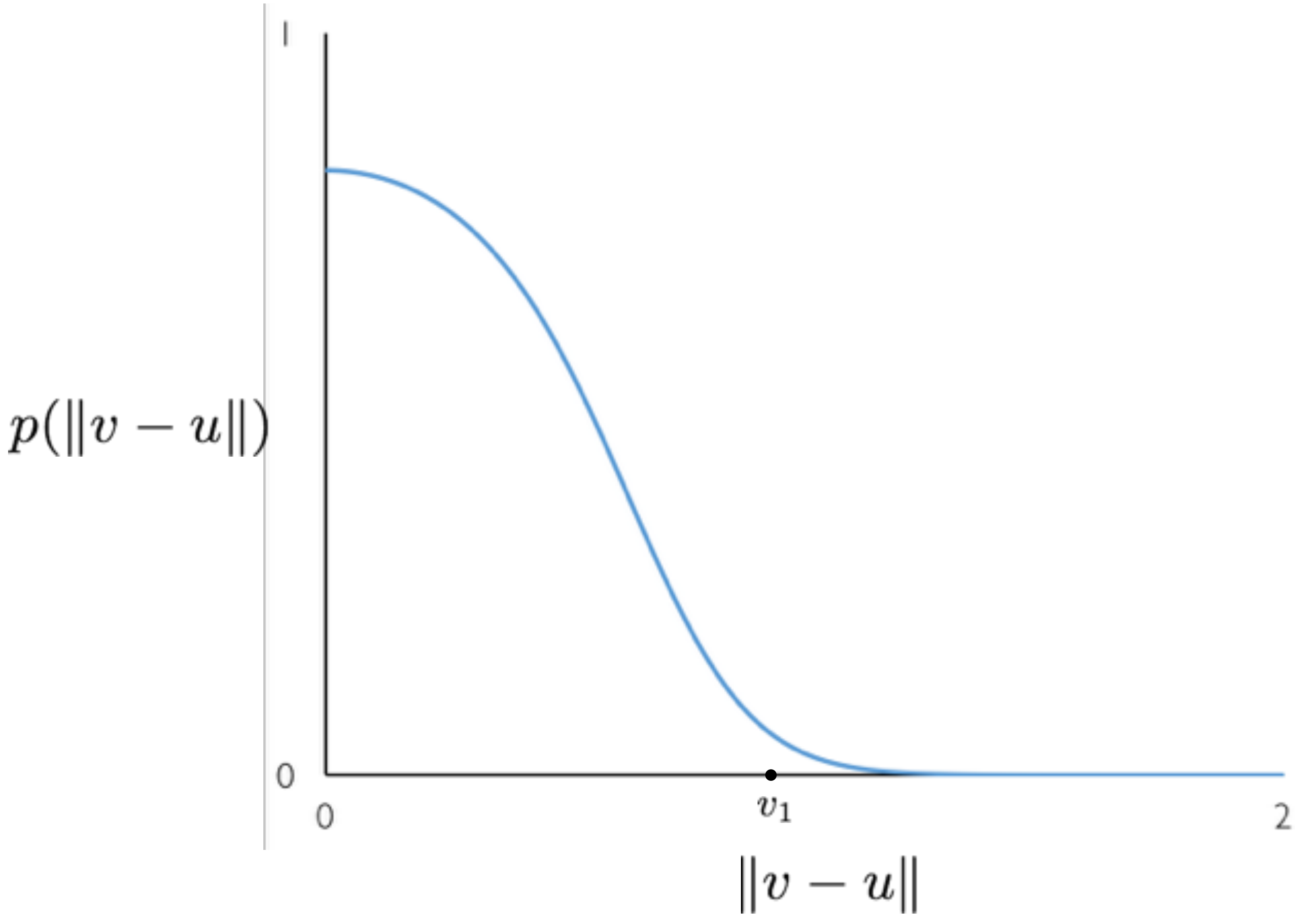
with expected amortized running time

$$\tilde{O}\left(\log(1/\epsilon)\, n^{2/3}\right)$$

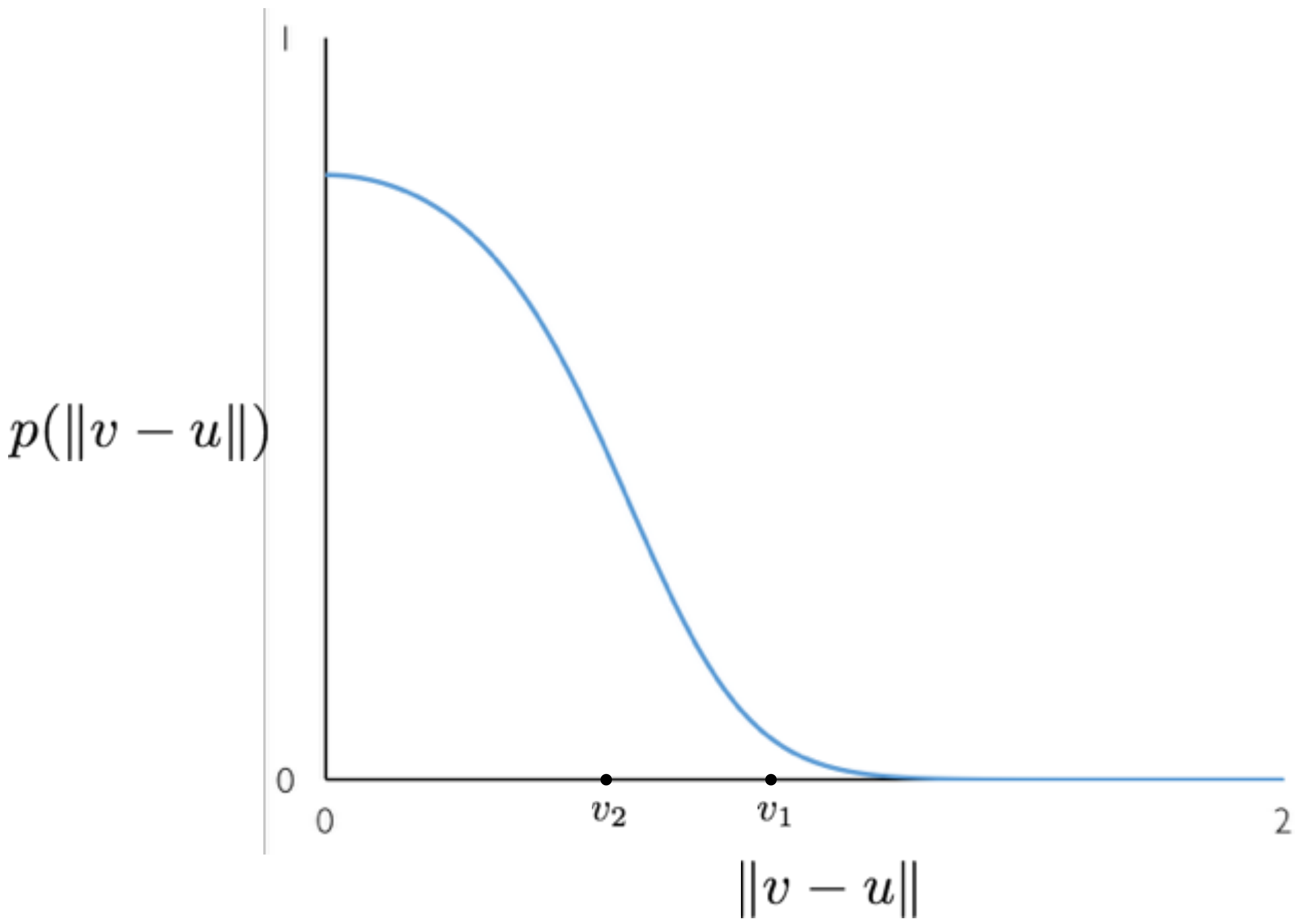
The Key Problem in Picture



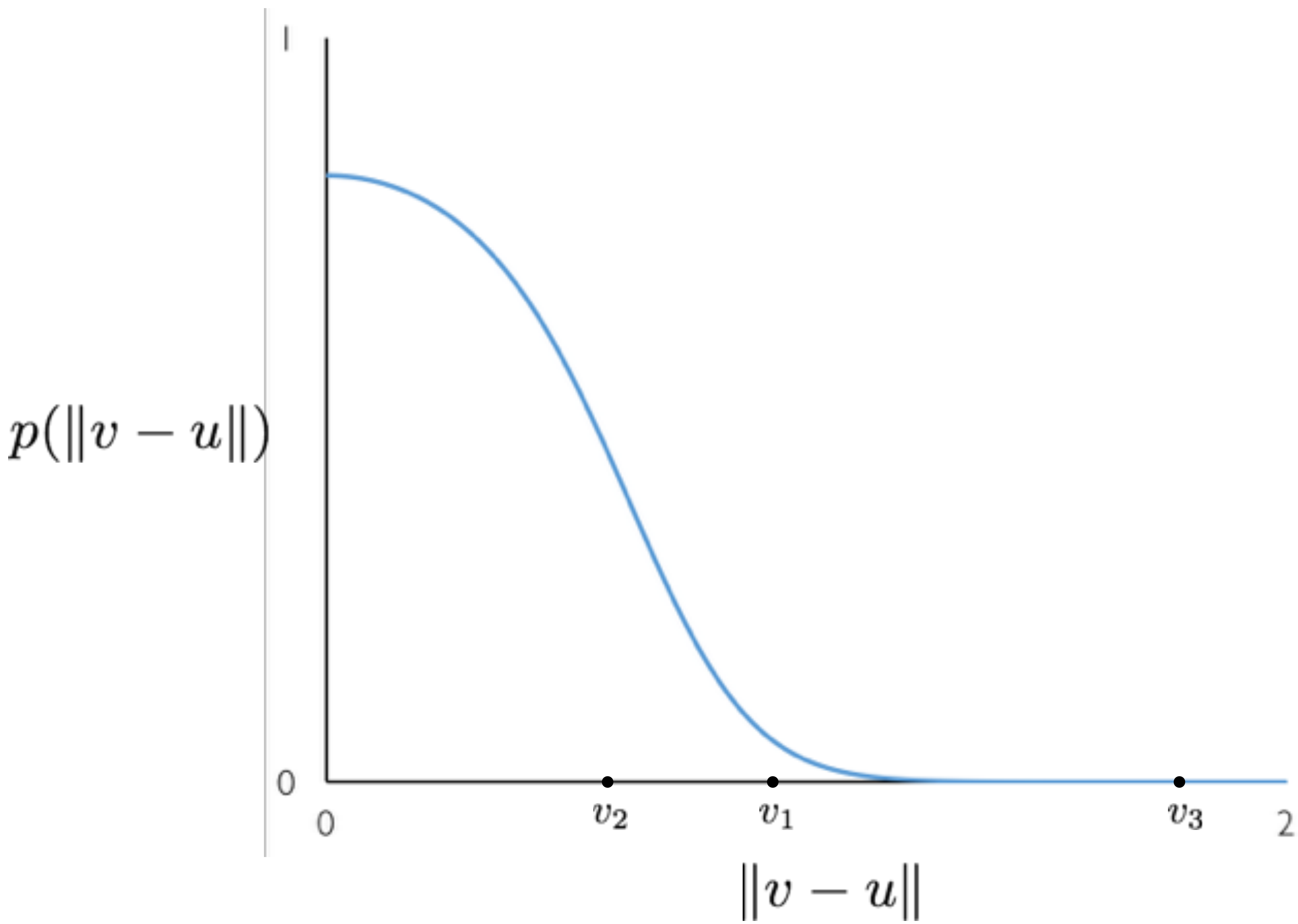
The Key Problem in Picture



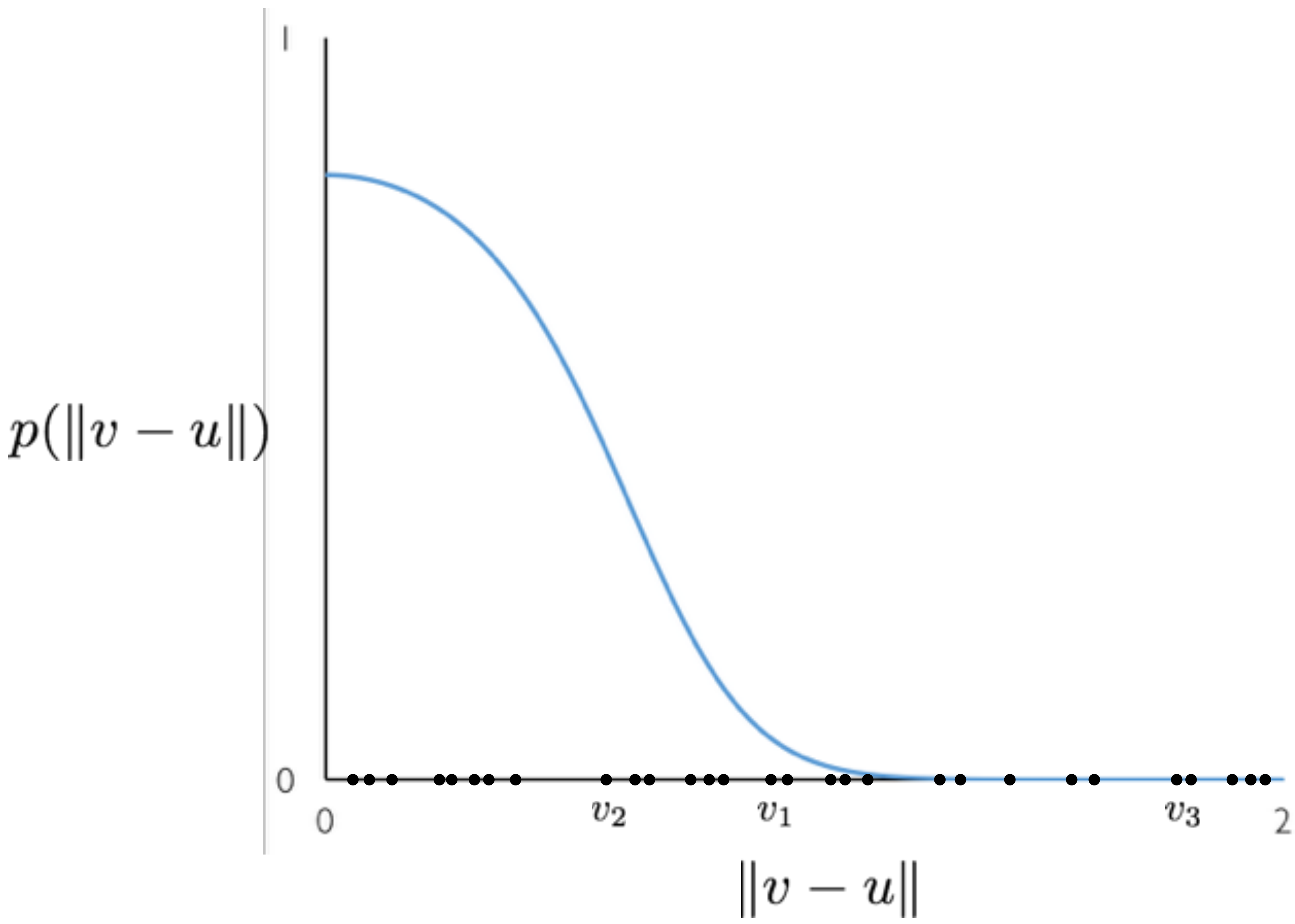
The Key Problem in Picture



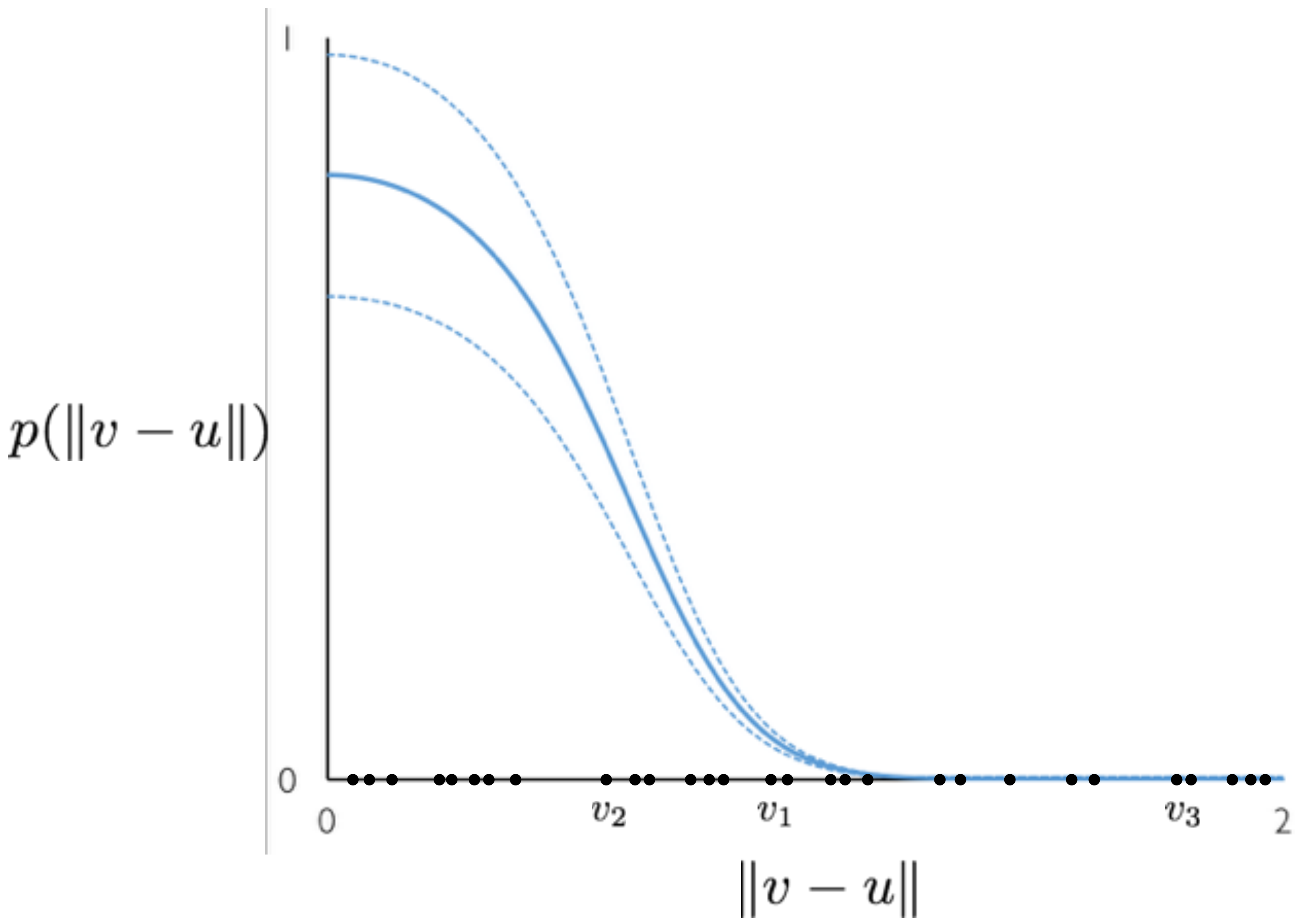
The Key Problem in Picture



The Key Problem in Picture

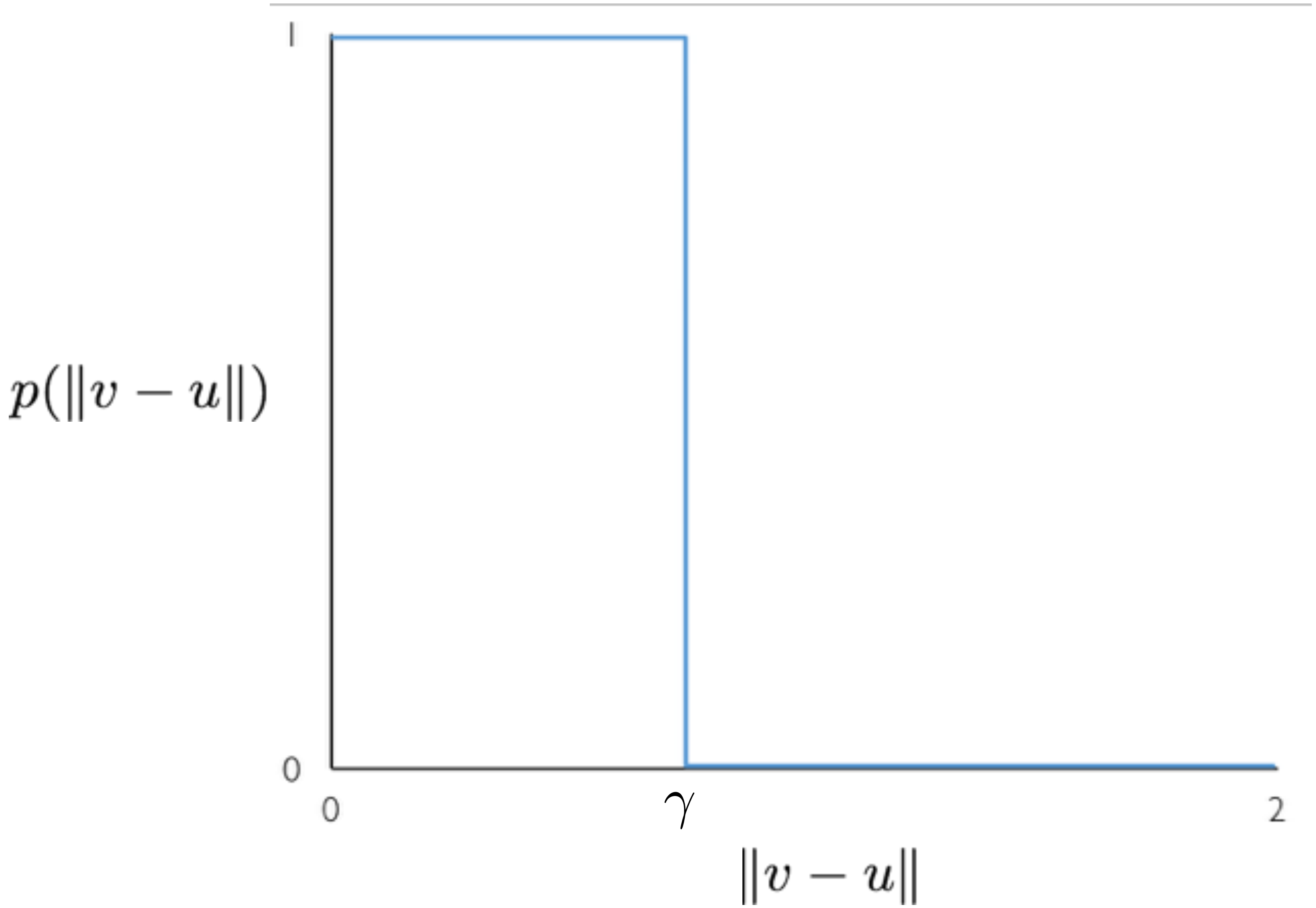


The Key Problem in Picture



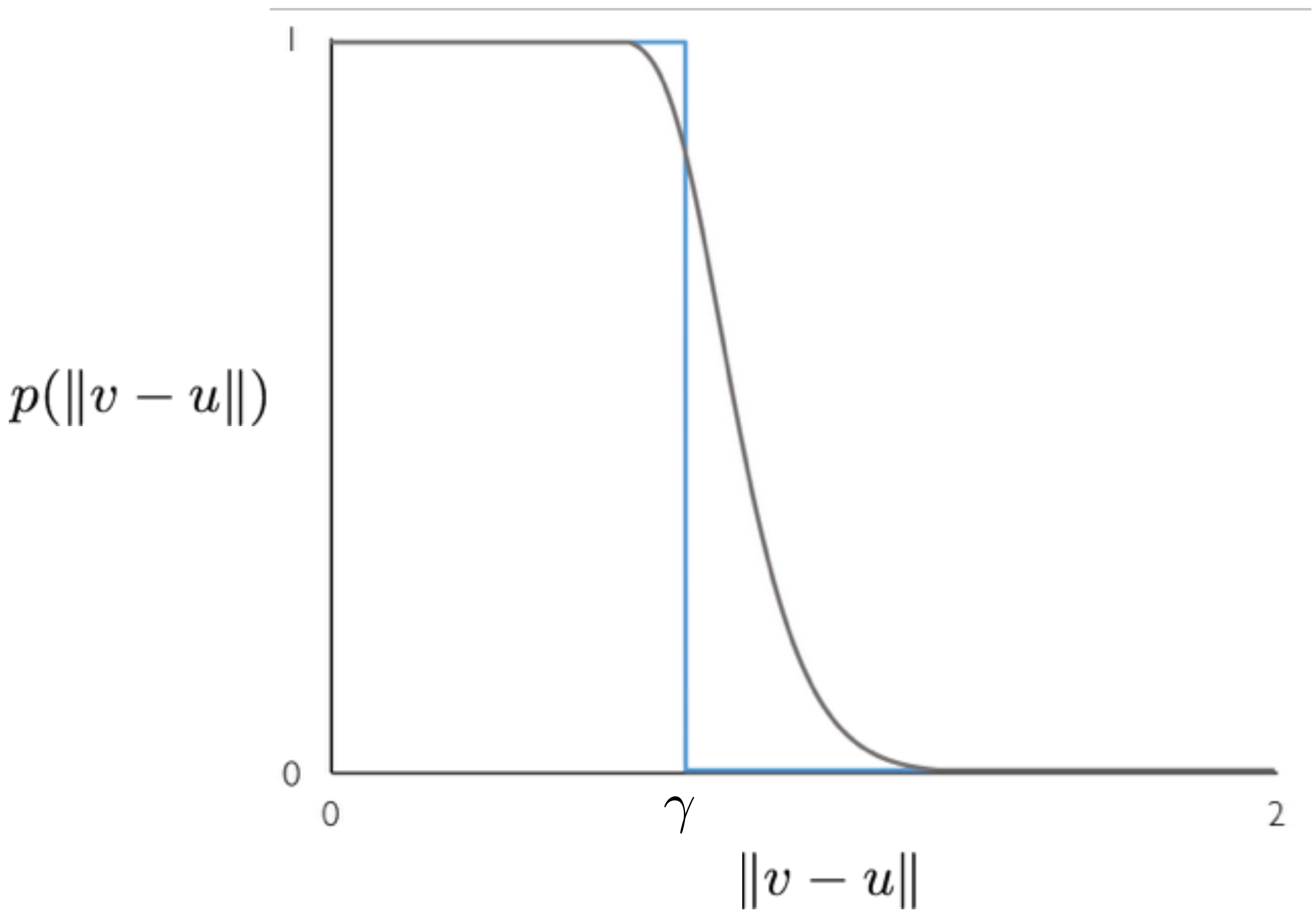
Example: Near Neighbor

- Return all $v \in \{v_1, \ldots, v_n\}$ s.t. $\|v - u\| \leq \gamma$



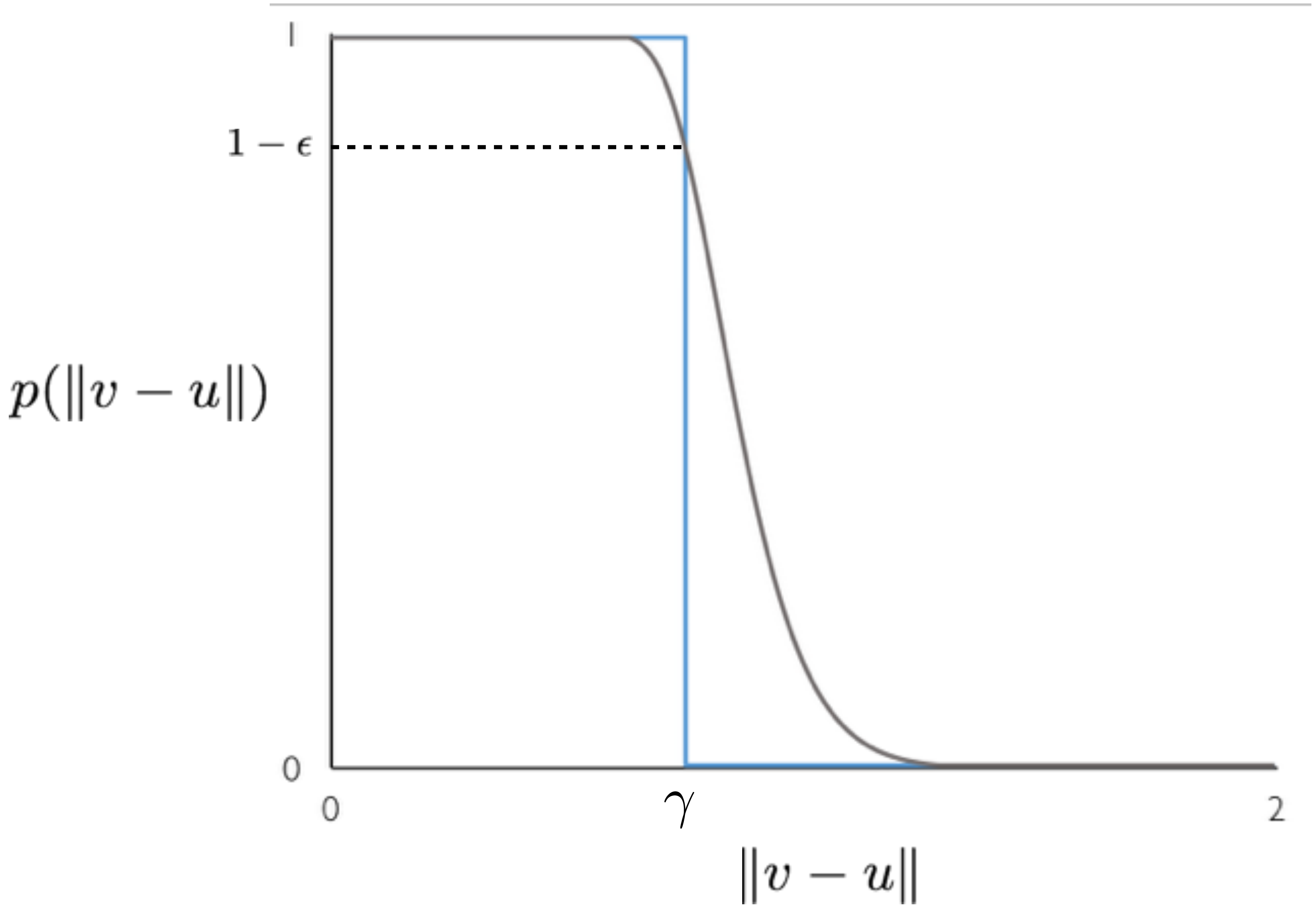
Example: Near Neighbor

- State of the art: Locality-Sensitive Hashing (LSH)



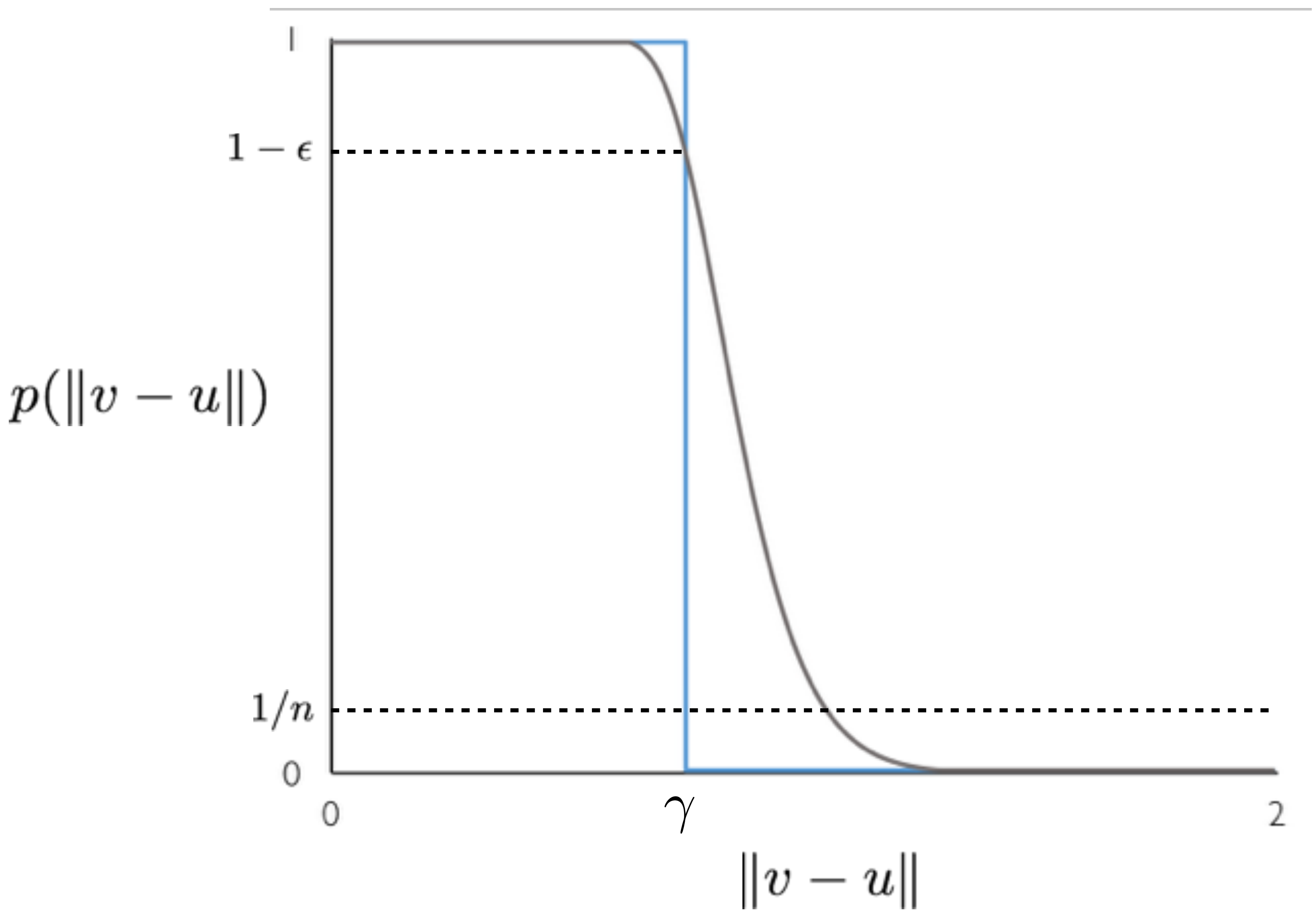
Example: Near Neighbor

- State of the art: Locality-Sensitive Hashing (LSH)



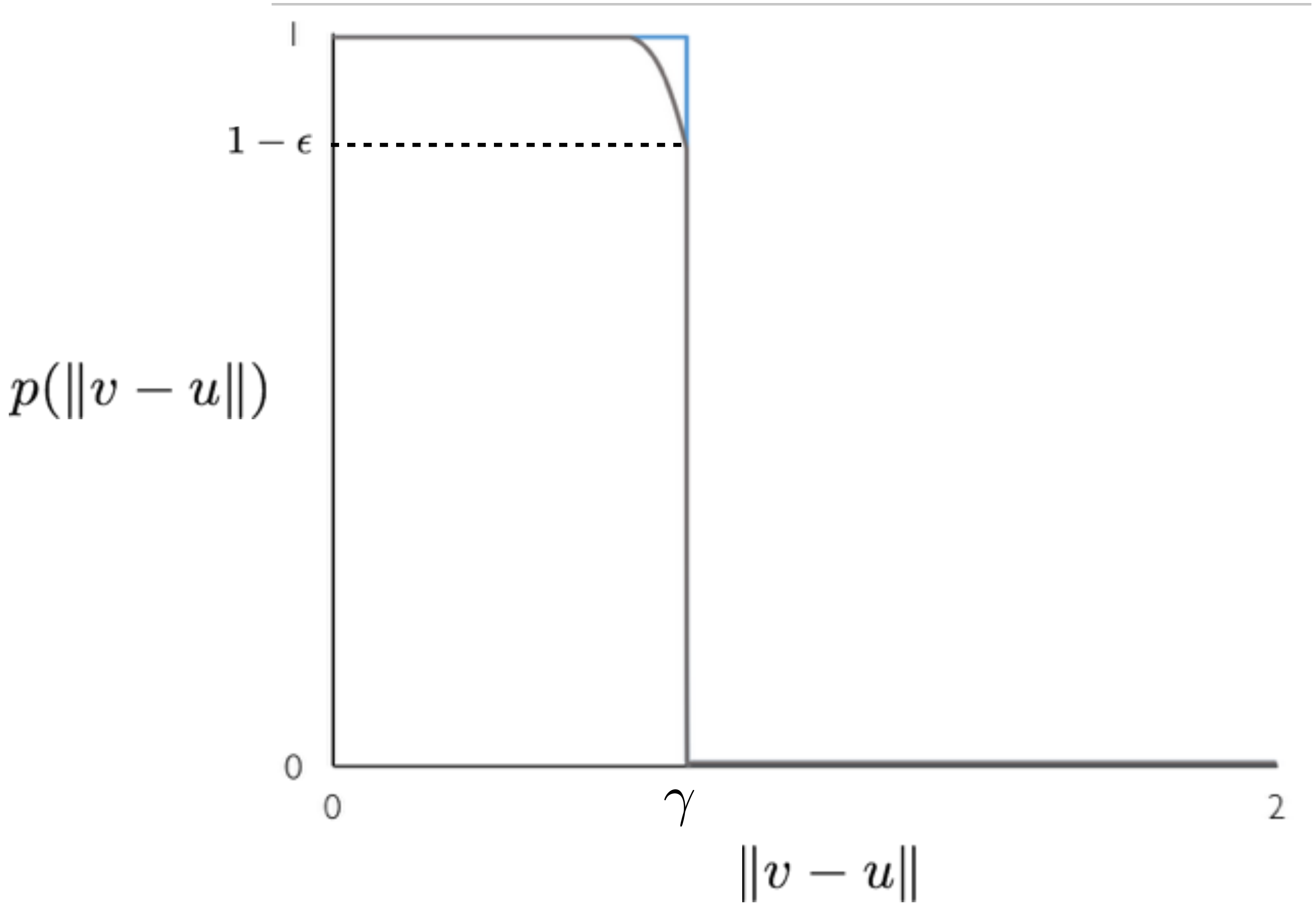
Example: Near Neighbor

- State of the art: Locality-Sensitive Hashing (LSH)

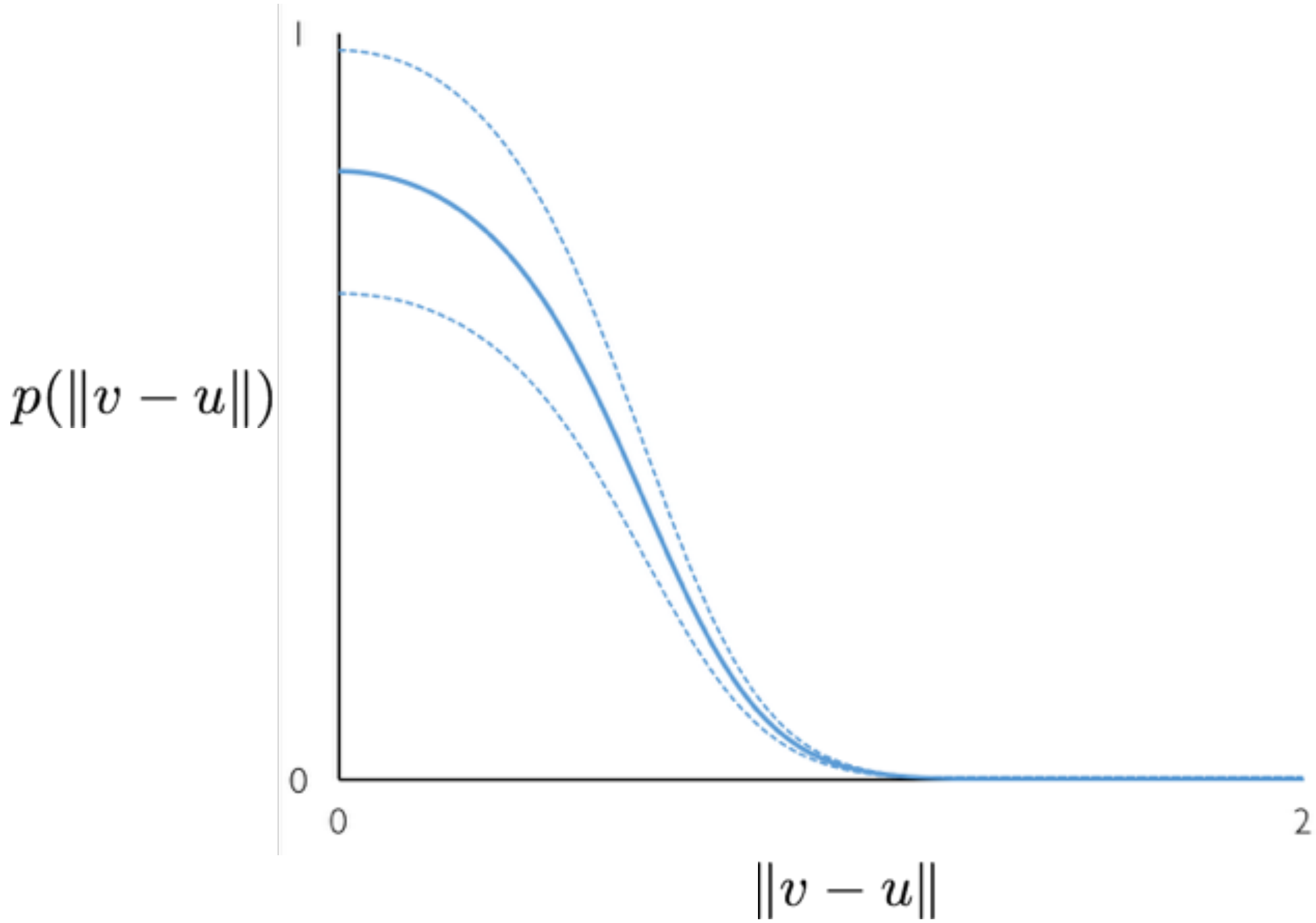


Example: Near Neighbor

- State of the art: Locality-Sensitive Hashing (LSH)

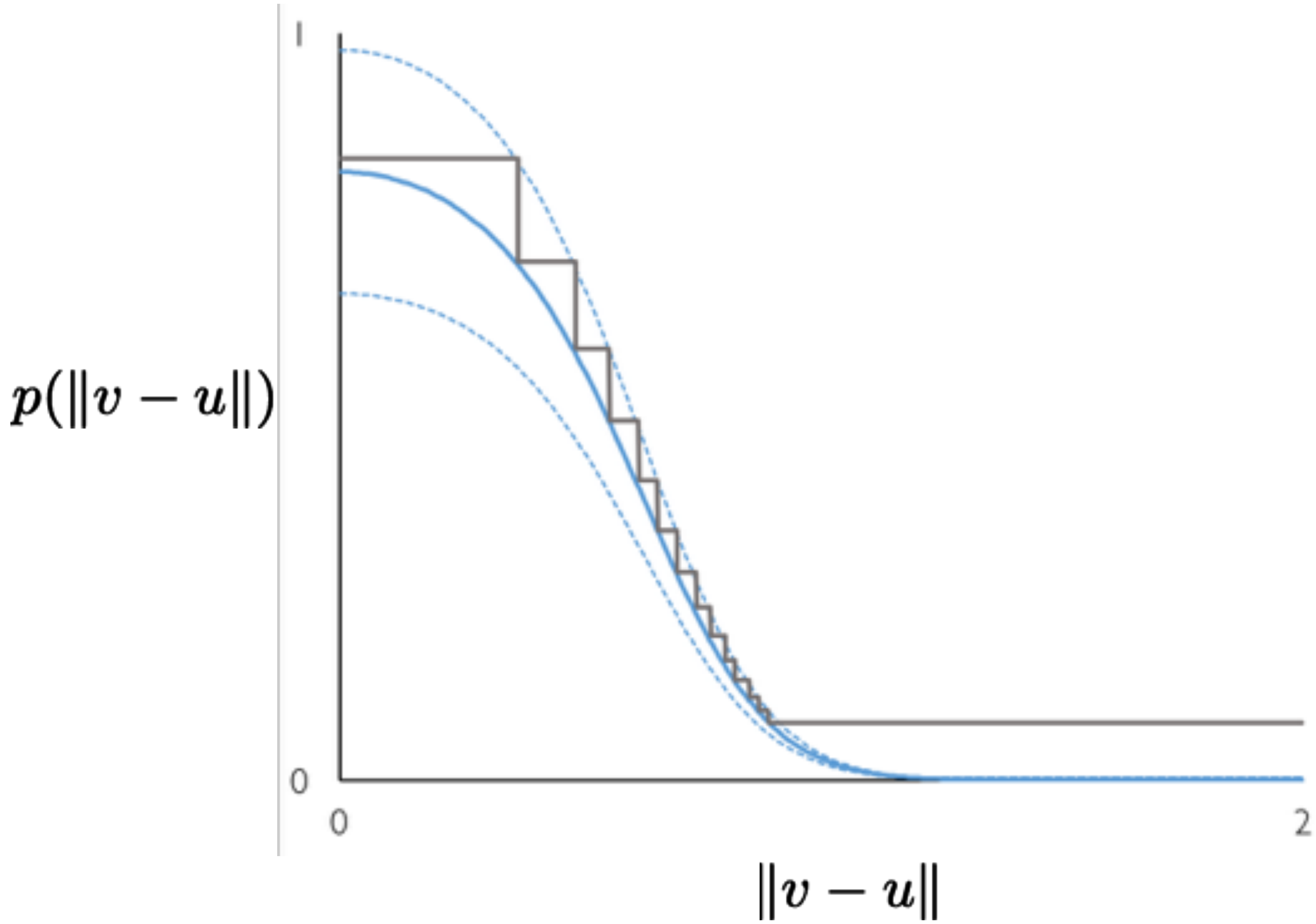


Locality-Sensitive Sampling



- We generalize LSH to arbitrary functions with only *constant-factor* increase in running time

Locality-Sensitive Sampling



- We generalize LSH to arbitrary functions with only *constant-factor* increase in running time

A Sublinear Result

- **Theorem [Farias, Li, S.]:** Under the stated assumptions, the greedy algorithm, combined with Locality-Sensitive Sampling, achieves in expectation

$$(1-\delta)(1-e^{-1})\mathrm{OPT}$$

with expected amortized running time

$$\tilde{o}_\delta\left(kn\right)$$

Experiment with Outbrain Data

Experiment with Outbrain Data

- In a month, Outbrain serves 250 billion personalized content recommendations

Experiment with Outbrain Data

- In a month, Outbrain serves 250 billion personalized content recommendations
- Dataset contains sample of pages viewed by users on multiple publisher sites in US between 14 - 28 June 2016
    - ~ 2 billion page views, 700 million unique users

Experiment with Outbrain Data

- In a month, Outbrain serves 250 billion personalized content recommendations
- Dataset contains sample of pages viewed by users on multiple publisher sites in US between 14 - 28 June 2016
  - ~ 2 billion page views, 700 million unique users
- Construction of embedding:
  - For each user, make chronological list of page views
  - Construct embedding of articles in $\mathbb{R}^{100}$ using word2vec

Structure of Experiment

Structure of Experiment

- Consider task of making 200 recommendations for each test user

Structure of Experiment

- Consider task of making 200 recommendations for each test user
- First 10 page views are taken as the samples from the distribution U

Structure of Experiment

- Consider task of making 200 recommendations for each test user
- First 10 page views are taken as the samples from the distribution U
- Compared the performance of LSF with two benchmarks:
  - *Last viewed* - Recommends nearest neighbors of the last viewed page
  - *Mean* - Recommends nearest neighbors of the mean of the samples

Experiments: Results

Experiments: Results

Algorithm	Hits	Improvement

Experiments: Results

Algorithm	Hits	Improvement
LSS	3.6%	

Experiments: Results

Algorithm	Hits	Improvement
LSS	3.6%	
Mean	2.8%	28.5%

Experiments: Results

Algorithm	Hits	Improvement
LSS	3.6%	
Mean	2.8%	28.5%
Last viewed	1.6%	125%

Experiments: Results

Algorithm	Hits	Improvement
LSS	3.6%	
Mean	2.8%	28.5%
Last viewed	1.6%	125%

28.5% improvement corresponds to revenue increase of $\sim$ \$68 million

Thanks!

LSH

- LSH is a hashing scheme such that nearby points are more likely to have same hash value
- LSH algorithm:
  - Pre-process points in search space to obtain their hash values
  - Find hash value of the query and return input points having same
- LSH algorithm can determine if there exists a near neighbor with some probability in sub-linear time

Details on Near Neighbor

$$\hat{v} = argmax_{v \in V} \frac{1}{s} \sum_{i=1}^{s} \mathbf{1}(v^T u_i > \gamma)$$

- Reframe the problem as Near Neighbor queries
- Define Near Neighbor $NN(u) = \{v : v^T u > \gamma\}$
- Search space for $\hat{v}$ is narrowed from V to $\cup_{i=1}^{s} NN(u_i)$
- If the NN queries run in sublinear time and size of the set $\cup_{i=1}^{s} NN(u_i)$

  is sublinear, then $\hat{v}$ can be calculated in sublinear time

  - NN queries can be answered approximately in sublinear time using Locality Sensitive Hashing (LSH)